Problem 7 Editorial: nasin kalama pi toki pona
6+2=8 Points

Problem ID: `toki`
Rank: `2+3`

Special thanks to Jordan Chong of team The Phoenixes for contributing this editorial and allowing us to use his Python code!

Overview

Problem statement: https://calico.berkeley.edu/files/calico-fa22/contest/toki/toki.pdf

This problem features toki pona, a real constructed language (conlang) inspired by minimalism—which Ryan (“alfphaderp”) and Zaid (“jan Semaja”) from the CALICO team both know! What does the title mean? Words in toki pona have many different meanings, but the title approximately translates to “the phonotactics (‘way of pronunciation’) of toki pona.”

The main test set has words with a syllable count sufficient for a hard-coded solution. The bonus has words long enough to require a generalized solution—different paths like dynamic programming and simplifying the problem are some possible approaches—all running in linear time. The bonus also featured a bounty for lowest code size to solve the test set (in bytes).

If you want to learn toki pona, you can get the official book, the official dictionary, or learn it for free with 12 Days of sona pi toki pona by jan Misali or lipu sona pona by /dev/urandom.

Main Test Set

This test set contains words with up to six letters, with valid words consisting of at most two syllables. As a result, we can do some casework to identify valid words. Note that throughout this editorial, all values are zero-indexed and "illegal sequences" include adjacent vowels.

Hard Code *Solution available in Python (solution provided by Jordan Chong)*

To avoid complications, we can first search through $\mathbf{W}$ for illegal letters and sequences. We can do this by creating an index i that starts at 0 and increments until it reaches the length of $\mathbf{W}$. At each step, check that $\mathbf{W}[i]$ is part of the toki pona alphabet and that $\mathbf{W}[i] + \mathbf{W}[i+1]$ is not an illegal sequence. If no illegal letters or sequences are present in $\mathbf{W}$, we can continue to check its syllable structure.

Since every word in this test set has at most two syllables, our main challenge is finding the character position p at which the second syllable starts.

If $\mathbf{W}[0]$ is a consonant, $\mathbf{W}[1]$ must be a vowel. If the word's length is three or more, $\mathbf{W}[2]$ must either be `n` or another consonant ($\mathbf{W}[2]$ can't be a vowel, as we've scanned for double-vowels in the previous step):

1. If $\mathbf{W}[2]$ is `n`, we can safely consider $\mathbf{W}[2]$ to be part of the first syllable. This is because if the next letter is a consonant, this `n` must be part of the first syllable; if the next letter is a vowel, it doesn't matter which the `n` belongs to. Following this logic, $p = 3$.
2. If $\mathbf{W}[2]$ is a non-`n` consonant, it must be the start of the second syllable, so $p = 2$.

If $\mathbf{W}[0]$ is a vowel, the logic is largely the same as the above case, but with all indices offset backwards by one: if $\mathbf{W}[1]$ is `n`, $p = 2$, and if it is a non-`n` consonant, $p = 1$.

The second syllable thus consists of all characters after index p, inclusive—unless p is equal to the length of $\mathbf{W}$ (in which case we can stop as the word contains only one syllable). If its first letter is a vowel, we must check that the next letter either is `n` or doesn't exist. If its first letter is a consonant, we must check that the next is a vowel and that the third is `n` or doesn't exist.

This approach has a linear $O(|\mathbf{W}|)$ time complexity arising from the search for illegal letters and sequences—although, this may be a bit misleading as the solution is unable to correctly evaluate words with more than two syllables.

Bonus Test Set 1

With up to 10^3 = 1000 letters in $\mathbf{W}$, there can be as many as 500 syllables! The need for a flexible, generalized solution opens the door to a variety of different approaches. This problem also featured a bounty: the two teams with the smallest code (in bytes) that solved this test set won a free copy of Toki Pona: the Language of Good by Sonja Lang.

Generalization *Solutions available in Java, Python (solution provided by Jordan Chong)*

We can generalize our approach from the main test set for words with more than two syllables. We start by searching for illegal letters and sequences, then execute the approach for finding the first syllable. At this point, it might be easier to create a function that evaluates syllables; we can deal with a potential trailing `n` by evaluating $\mathbf{W}[p] + \mathbf{W}[p+1] + \mathbf{W}[p+2]$ first, then $\mathbf{W}[p] + \mathbf{W}[p+1]$, and then $\mathbf{W}[p]$. Once p is found, we can simply repeat the process until we reach the end of the word and $p = |\mathbf{W}|$. This approach's time complexity is also $O(\mathbf{N})$ as it loops through $\mathbf{W}$, but can work for words of many syllables.

Dynamic Programming *Solutions available in C++, Python*

If we want to avoid the ambiguity of when one syllable ends and the other starts, we can recursively check if assuming the next syllable in $\mathbf{W}$ being of length one, two, or three leads to a valid word being constructed. To avoid complications, we can first search through $\mathbf{W}$ for illegal letters and sequences.

One can note that $\mathbf{W}$ is only valid if its first syllable is valid and the rest of the word (following the first syllable) is, too. We can thus recursively check the rest of the word the same way until the last syllable is reached. Let's create a function dp where $dp(i)$ returns the validity of the suffix of $\mathbf{W}$ starting at i ('pona' = True). Our end goal is for $dp(0)$ to contain our answer. At the moment, we don't have any $dp(i)$ values at our disposal. However, note that the index $|\mathbf{W}|$ is the index after all letters in the word. Thus we can set $dp(|\mathbf{W}|)$ to return True, and continue backwards—evaluating $dp(i)$ for decreasing values of i until we reach $dp(0)$.

For $dp(i)$, we can set it to True if:

1. The next substring of length 1 is valid and $dp(i+1)$ is True, or
2. The next substring of length 2 is valid and $dp(i+2)$ is True, or
3. The next substring of length 3 is valid and $dp(i+3)$ is True

Once we've filled in all dp values from $|\mathbf{W}|$ to 0, $dp(0)$ will give us our answer.

The initial step of checking for illegal characters take $O(|\mathbf{W}|)$ time. Each dp value takes constant time to evaluate, so evaluating $|\mathbf{W}|$ values takes $O(|\mathbf{W}|)$ time as well. As a result, this solution has an overall runtime of $O(|\mathbf{W}|)$.

pali pona *Solution available in Python (solution provided by Jordan Chong)*

We can simplify our implementation by trying to find a common theme behind illegal syllable structures (assuming no illegal sequences are present). For the following examples, let $\hat{C}$ denote a non-`n` consonant:

1. $(C)V\hat{C}$: for this to be relevant, the next syllable must start with a consonant—otherwise, the syllables could be $(C)V$ - $\hat{C}V($`n`$)$, which is valid. Here, there are **two adjacent consonants where the first is not `n`** ($\hat{C}$ from this syllable + the first consonant of the next syllable). In the event that there is no next syllable, this syllable structure would still be invalid—we'd need to check for this separately.
2. $\hat{C}$`n`: this syllable is missing a vowel in between the two characters. Here, there are also **two adjacent consonants where the first is not `n`.**
3. $\hat{C}$: the first case actually can be reduced to this one, as you could split $(C)V\hat{C}$ up into $(C)V$ - $\hat{C}$! As such, the logic here is the same as above.
4. `n`: for this to be relevant, there can't be any vowels to the left or right of `n` to tag on to. This means that the previous syllable not only ends with $\hat{C}$ (making itself also invalid, see #1), but we again have **two adjacent consonants where the first is not `n`** (the $\hat{C}$ from the previous syllable + the `n` from this one). In the event that there is no previous syllable, this structure would still be invalid—we'd also need to check for this separately.

We can deduce that having **two adjacent consonants where the first is not `n` is a property unique to invalid syllables**. As a result, we can just check that $\mathbf{W}$ only contains letters from the toki pona alphabet, has no illegal sequences, and has no adjacent consonants where the first is not `n`.

There are, however, two edge cases as mentioned above. For $\mathbf{W}$ to be valid, we have to make sure that it does not end in a non-`n` consonant (see #1). Also, if $\mathbf{W}$ starts with `n`, we must check that the second letter is a vowel (see #4). This solution, like the others, runs in $O(|\mathbf{W}|)$ time as it indexes through $\mathbf{W}$.

Regex *Solutions available in Java, Python*

The most effective way to codegolf this problem is to use regex (regular expressions), a way of searching through strings like $\mathbf{W}$. Regex operations are often defined by single characters, reducing code size and allowing solutions to be only one or two lines long. Codegolf solutions vary, but we can try to find the lower bound of characters for a solution (Kolmogorov complexity).

We assume Python usage and that the theoretically shortest solution uses regex.

- `import re` (regex) and a line break: 10 chars
- `for _ in[*open(0)][1:]:` (takes $\mathbf{T}$ and loops over it): 23 chars
- `input()` to read $\mathbf{W}$ from standard input: 7 chars
- `[wu,wo,ji,ti,nn,nm]` stores illegal sequences (brackets signify a container): 19 chars
- `[aeiou]` at least once: 7 chars
- `[j-nptsw]` stores consonants (`j-n` is a way of storing `jklmn` using regex): 9 chars
- `[pona]` and `[ike]`: 11 chars (alternatively we can use `[pioknea]`, a combination of "pona" and "ike," but this would require other operations that contribute characters)
- `print()` to write to standard output: 7 chars

This equals **93 bytes** (1 character = 1 byte), but this is a low estimate. We don't consider necessary operations on $\mathbf{W}$ because algorithms used can vary. The contest's two lowest byte counts were from teams Grizzlies (205 bytes) and Placeholder (230 bytes), trailed very closely by the runner-up team RIP Poggers (232 bytes)!